viewpoint

The Fate of Fortran-8x

A new ANSI/ISO standard for Fortran seemed to be imminent in October, 1987, when the draft specification for the new language (provisionally called Fortran-8x) was published for public review, this being the formal ANSI procedure prior to adoption of a new standard. During the four-month public-comment period (October 23, 1987–February 23, 1988) I read the specification (a 340-page book designated S8.104)[1] and sent my 61-page Comment to the ANSI drafting committee (X3J3) in February, 1988.

My comment was broadly critical of the specification for reasons that are summarized here, and I was invited to present my views to the X3J3 committee at its meeting in Jackson, Wyoming, in August, 1988. Attendance at the week-long meeting was educational. I learned that the X3J3 committee consists of about 40 volunteers representing the major computer vendors and Fortran users.

The X3J3 meetings had the flavor of a political forum dominated by debate about procedure and agenda with frequent votes on alternative proposals. The kind of technical discussion that one would expect in a language design committee played a very small part in the proceedings. The chair of X3J3, Jeanne Adams, acted mainly as a mediator between the four or five factions within the committee; Fortran-8x only reached the public-comment stage after compromises within X3J3, as evidenced by *Removed Extensions* in S8.104.

This dissension within X3J3 arises from a deep-seated technical incompatibility. Fortran-77 and earlier dialects are fundamentally flawed by static memory allocation (as will be explained), and some members of X3J3 clearly recognize that in order to change it into a modern, well-designed language, one must not only introduce new features, but also remove some old features because old and new cannot rationally co-exist. This is explicitly recognized in S8.104 in a list of *deprecated* and *obsolescent* features.

The countervailing argument is that the new language must be upward-compatible with the old language. The S8.104 specification attempted to reconcile these conflicting purposes through the concept of *Language Evolution*, whereby redundant and obsolescent features would be slowly removed from the ANSI standard over a 20–30-year period. However, at the August 1988 meeting the concept of Language Evolution was abandoned as one more compromise between desirable change and upward compatibility.

Following several more X3J3 meetings, version 112 of the specification was put out for public review in 1989 (July 27–November 24). The revisions since the first public review are (from my standpoint) minor; they do not address the fundamental problems that formed the basis for my criticism and recommendations.[2]

Synopsis of Recommendations

The purpose of the summary is to re-affirm my technical judgment that Fortran-8x should *not* become an ANSI Standard regardless of further (minor) revisions to the specification. Fortran-77 should be retained as the *last* ANSI/ISO Standard dialect,[3] and its deficiencies should be remedied by selection of an established block-structured language (probably ADA, possible Algol-68) as an alternative to Fortran-8x. Fortran programmers should be advised to switch to this alternative language for their future programming projects. Translation of existing programs should be automated with a Fortran-77 to ADA/Algol-68 pre-processor.

This strategy would bring U.S. practice up to the state of the art in the U.S.S.R., where Algol-68 is both the dominant programming language and the architectural basis of their mainframe computers. While similar, technically well-designed systems are available in the U.S. (on the Unisys/Burroughs mainframes), they are not widely used.

My overall conclusions about Fortran-8x are as follows:

- Its semantic implications are not fully defined in S8.104 especially in regard to the introduction of dynamic memory allocation. This is the *implicit* new semantic feature of Fortran-8x that makes it radically different from all earlier dialects of the language.
- Fortran-8x will be a vast and complex language similar to PL/1. The creation of fully defined, bug-

[1] i.e., the version of the specification for Fortran-8x current at the 104th meeting of X3J3.

[2] Despite ANSI's mandate, X3J3 has not formally responded to my public comment.

[3] ANS Fortran-77 was re-affirmed by the ANSI Board of Standards Review on August 1, 1989. Furthermore, on October 18, the ANSI X3 committee voted to retain Fortran-77 as a distinct standard regardless of the eventual fate of Fortran-8x.

free compilers will be difficult and costly.
- The plethora of different concepts and elements of the language, each with its own rules, exceptions, prohibitions, and constraints, will make programming in Fortran-8x a highly error-prone, laborious, and costly business. Many of the elements are redundant, and some of the concepts are logically wrong.
- The introduction of Fortran-8x would be contrary to the trend in computer systems, i.e., towards simple, orthogonal designs based on a minimal set of essential concepts.

Politics, History, Commerce, and Technology

My recommendations will surprise the many people who accept Fortran as the *lingua franca* of scientific and engineering programming. In view of its widespread use there must be very well-founded reasons for abandoning Fortran in favor of an alternative language. These reasons are best appreciated in the context of the history of computing, presented here from a personal perspective.

I wrote my first computer program in 1961, and in 1962 I began using a new computer (the Manchester Atlas) that implemented dynamic memory allocation via a hardware stack. Its algebraic programming language, Atlas Autocode (an Algol-60 dialect) had nested block structure, dynamic array dimensioning, and recursive routines, features that I recognized then as vitally important and whose fundamental significance was revealed to me some years later after studying the theory of computation.

Yet, Fortran programmers will have to wait until the 1990s for these vital programming tools—until ADA of Algol-68 supersedes Fortran-77. You may well ask (as I have been asking for the past 25 years) why has it taken so long? And furthermore, what are the likely consequences for Fortran programmers?

The brief answer to the "why so long" question is that IBM adopted the fundamentally flawed design strategy of static memory allocation for the System/360, and as a result of their dominant influence on the whole industry, the design flaw was propagated into many other manufactuers' hardware and software. The flaw became a defacto "standard," whose detrimental effects on the art and science of computing seem destined to last well into the 21st century.

The Fundamental Theory

The constraint of static memory allocation is a fundamental flaw because the most general kind of computational process (a Turing Machine process) requires dynamic memory allocation. This requirement distinguishes the Turing Machine from the Finite-Automaton in the theory of computation; the latter is characterized by static memory allocation.

Every process involves a program and data, the essential distinction between them being that the program is the fixed (invariant) part of the algorithm while the data is different in each run of the program. The split between the program part and the data part depends on the algorithm's specification.

Any process whose specification implies a variable amount of data storage is a Turing Machine process, and if the program is to have its invariant attribute, then the programming language and hardware must allow for dynamic allocation of memory for the variable amount of data.

The vast majority of computing tasks are Turing Machine processes, i.e., they demand dynamic allocation of memory. For example, a typical scientific/engineering application has at least one array whose size is data dependent. The array declaration in the program should contain a run-time generated number as its size, and this should compile into run-time (i.e., dynamic) allocation of the requisite amount of memory for the array. Recursion, which has an essential role in the theory of computation, also requires dynamic memory allocation—for the stack of return addresses and for the local variables of each recursive call.

If the programmer is restricted to a language that does not allow for dynamic memory allocation (as I was restricted to Fortran in 1965–75), then he or she must inevitably make the program data-dependent, thus vitiating the essential distinction between program and data.

When is a Program not a Program? —when it is written in Fortran.

The practical consequence of this fundamental flaw in the Fortran language is that the user of the program finds him or herself having to revise and recompile the program to match the data that is being processed. In cases where array dimensions are necessarily calculated at run-time, the programmer may find him or herself "inside a loop" (from personal experience), editing and recompiling the program until he or she iterates (by trial and error) to array dimensions that match the data.

Thus, the static-allocation design-philosophy underlying Fortran causes inefficiency (in both machine and programmer time) compared with dynamic allocation systems such as Algol on a Burroughs mainframe.

After 30 years (1957–87) of unwittingly laboring under the debilitating constraint of static memory allocation Fortran programmers are to be released from their shackles. Fortran-8x or preferably ADA/Algol-68 will have dynamic array dimensioning, recursive routines, and nested block structure. However, Fortran-8x (unlike the preferable ADA/Algol-68 alternative) would limit the nesting to two levels—one of its rational compromises.

Block structure, which originated in Algol-60, is fundamentally important for localizing environments and for achieving communication between a subroutine and its calling environment. A detailed exposition is appended to my comments on Fortran-8x.

Fixing the Flaw vs. a Fresh Start

Despite the inclusion of dynamic memory allocation facilities, the prospect offered by Fortran-8x is not an attractive one. The overall prob-

lem is that the X3J3 committee is trying to reconcile two conflicting purposes:

(1) The desire for compatibility with Fortran-77 so that existing programs will compile.
(2) The need to enhance the language with new features:
   - to correct the design flaws that were built into Fortran 30 years ago, and
   - to add features (such as array-returning functions) that have been available elsewhere (i.e., in Algol-68) for 20 years or more.

The complexity of Fortran-8x arises from adding a large collection of new language elements while retaining all the old ones. Conflicts and redundancies between old and new are inevitable. For example, there are six different kinds of array declaration in Fortran-8x, and yet analysis shows that only two are essential: one to allocate memory for an array and the other for array parameters of subroutines. Even if the other four were deleted from Fortran-8x (in the year 2021, according to X3J3's schedule for language evolution—prior to its abandonment of evolution), the syntax of the remaining two should be changed to reflect their very different semantics.

The features of the S8.104 specification illustrate its overall lack of coherence and rationality: the SAVE and RECURSIVE declarators. SAVE was introduced into Fortran-77 as a semantically insignificant feature. The Digital/VAX Fortran-77 manual says that SAVE is "redundant" to their compiler—because everything is always saved, i.e., statically allocated. In Fortran-8x, where dynamic memory allocation must be implemented, the absence of SAVE declarations in old and trusted "proven" programs will cause them to fail, notwithstanding X3J3's claim that any Fortran-77 program will conform under the Fortran-8x specification. I named this phenomenon **The SAVE Time Bomb** because it will likely cause explosions throughout the Fortran-speaking would.

While Fortran-8x will allow recursive routines, the S8.104 specification says that these must be qualified with the RECURSIVE declarator. If subroutine call and return is implemented by the well-known mechanism of using a stack for the return addresses, there is no penalty in CPU time and memory usage (compared with alternative mechanism), and the bonus of recursion is obtained automatically, i.e., without any modifications (to allow for recursion) to either the parser or the code generator.

Thus, the RECURSIVE declarator is totally redundant. For this reason, programmers should not be faced with the needless task of deciding whether a subroutine is (indirectly) recursive or not. Hence, I concluded that the RECURSIVE declarator is a design error in Fortran-8x. Its total redundancy parallels that of SAVE in Fortran-77.

In Summary

Despite important technical enhancements, notably implicit Dynamic Memory Allocation,[4] the Fortran-8x proposal is unsatisfactory because the language is encumbered with many redundant and mutually incompatible features.

The only technically rational way of advancing the art of scientific and engineering programming is to abandon Fortran in favor of a modern, block-structured language such as Algol-68 or ADA.

Geoffrey Hunter
Chemistry Department
York University
4700 Keele Street
Toronto, Ontario
Canada M3J 1P3

In Response to the Fate of Fortran-8x

Geoffrey Hunter raises some interesting issues regarding the proposed ANSI/ISO Fortran standard, commonly known as Fortran-8x (although in 1990 the designation "8x" appears inappropriate!). My overall reaction to his objections is that the issues most important to him are not very important to the actual heavy-duty Fortran users, and the issues most important to the heavy-duty Fortran users are ignored by Hunter.

Hunter's central objection to the proposed Fortran standard is that it is not a modern block-structured language. Who ever said that it would be? If someone has an application that critically relies on a block-structured language, he or she is not likely to try to implement it using Fortran.

What is Fortran used for today? It is used primarily for large-scale scientific computations. Such applications typically have relatively simple subroutine trees and data structures, and performance is a major concern. The currently proposed design of the Fortran language is entirely adequate for accommodating such applications. An equivalent statement of Hunter's principal objection is that the proposed Fortran standard should be rejected because it would not be suitable for writing operating systems.

Among my colleagues here at NASA Ames there are a large number of highly expert programmers and scientists who write serious applications on our supercomputers. Many of them have completed courses in computer science, and most have significant experience using other languages. And yet, in discussions with them on possible enhancements to our Fortran language environment, I have yet to hear any-

[4] A major deficiency of the draft specification is its failure to *explicitly* recognize that some of the new features imply Dynamic Memory Allocation. This semantic shortcoming parallels Fortran-77's failure to explicitly recognize its implicit constraint of Static Memory Allocation.

one mention Hunter's primary concerns. Similarly, in discussions with others at large government laboratories and the like, I have heard expressions of concern about other features of the proposed Fortran standard, but I have never heard anyone mention Hunter's central objections.

There are other flaws in Hunter's arguments. I simply do not believe that it is realistic to hope that serious Fortran application programs could be automatically translated to Ada or Algol-68. Also, the language Algol-68 may have merit, but the fact that it is used in the U.S.S.R. is not a compelling reason to adopt it in the U.S.

One of Hunter's central objections, that the Fortran language is inherently based on static memory allocation, is not really true either in theory or in practice. For example, the Fortran compilers available on Cray supercomputers have featured stack allocation for data and stack-based subroutine calls for some time. His warnings about the SAVE Time Bomb are similarly misplaced—scientists running codes on our Crays have found the adjustment to using SAVE statements virtually painless.

It is unquestionably true that there are flaws in the proposed Fortran standard, and many of them are inescapable consequences of the fact that Fortran is an old language designed before the theory and practice of programming was well-understood. I, for one, am disappointed in the clumsiness of a number of the proposed features of Fortran-8x. Hunter's point that Fortran-8x is contrary to the trend of computer languages to be simple, concise, and elegant is well-taken. Hunter's concern about the potential difficulty of writing efficient and bug-free compilers for Fortran-8x is particularly valid.

However, there are some very important advantages to the proposed Fortran standard. In particular, its inclusion of array computation constructs is, in my view, a very important step forward and by itself outweighs most of the objections that have been raised against it. It is noteworthy that the "obsolete" Fortran language is the first major language to take this step—such constructs are not yet proposed as part of the standard for any "modern" language.

We in the large-scale scientific computation community are now moving ahead very seriously with plans to utilize the highly parallel, teraflops-class systems that will be available before the year 2000. In a fairly wide range of applications that we wish to map to these systems, the central time-intensive computations can be expressed easily as array operations. This class of computations certainly includes the large-grid PDE codes that are the mainstay of centers such as ours. The array constructs of the proposed Fortran standard are perfectly suited for this type of computation. They represent the first serious step to providing standard parallel programming constructs that can be efficiently supported across a variety of parallel systems, including both SIMD and MIMD designs.

Because of the delay in the adoption of the Fortran-8x standard, a number of manufacturers of advanced systems have already taken the step of implementing the basic array computation features into their compilers. Among the vendors who now support these features are Cray Research, Inc. and Thinking Machines, Inc. So far, however, programmers are reluctant to incorporate these constructs into their codes for the very valid reason that until these constructs are part of the Fortran standard, their codes may not be portable to other systems.

Hunter summarizes by saying "the only technically rational way of advancing the art of scientific and engineering programming is to abandon Fortran in favor of a modern block-structured language such as Algol-68 or Ada." This suggestion will simply not be taken seriously by the heavy-duty scientific computation community. These users simply cannot walk away from large application programs with 100,000+ lines of code. Also, this suggestion will not be taken seriously by those of us who are exploring highly parallel scientific computation, since at present, there is no prospect of standardizing array computation constructs in any major language except Fortran-8x.

A case can be made that scientists should be encouraged to consider other languages for writing new applications. At NASA Ames, for example, a number of the scientists are writing scientific applications in the C language. Also, vendors of heavy-duty scientific computers need to be encouraged to support alternative languages on their system. Further, standards committees working on other languages need to consider incorporating array constructs and possibly multitasking constructs in their designs. But for the foreseeable future, Fortran will indisputably be the major language for scientific computation. It is totally unrealistic to pretend otherwise.

David H. Bailey
NASA Ames Research Center
Moffet Field, CA 94035

Additional Thoughts

I would like to comment on two aspects of Geoffrey Hunter's commentary: the matter of the evolution of Fortran and the workings of the ANSI X3J3 technical subcommittee. But first, I would like to report that X3J3 has officially adopted the informal name "Fortran 90," and the language is expected to be approved this year as an international standard and also a U.S. national standard, barring bureaucratic problems.

When work began in 1978 to revise Fortran-77, many of us felt that

(*continued on p. 398*)

It is very difficult not to accept his first statement since computers have influenced every aspect of our life. It is equally difficult to see Dijkstra's point of view with respect to the second and third arguments. It is because, as pointed out by distinguished colleagues, his argument consists of many inconsistencies and misinterpretations.

The shallowness of his comments on software engineering and software maintenance can be strikingly seen by anyone knowledgeable of what these terms stand for. It is no doubt an accepted fact that program testing can only demonstrate the presence of bugs but never their absence. Dijkstra's argument, however, seems to imply that we can do away with testing once the formal proof techniques are available to us. But is it not naive to assume that formal proofs will be error free? Formal proofs do need human interaction. And to err is human. Hence, as long as any technology has a human element associated with it, there is always a possibility of making errors. Hence, software testing and formal proof techniques can only be complementary, but can never be alternative to each other; and they together help reinforce user's confidence in the software.

Similarly, Hamming points out the fallacious reasoning of the dominos example, and Cohen points to the inconsistency of Dijkstra's argument with respect to the "power of down-to-earth mathematics."

As every one agrees, despite inconsistencies and misinterpretations, Dijkstra's suggestions are worth looking at. His prime suggestion that formal methodologies need to be taught right from the introductory computer courses is one of them. It would be programmer's delight to have formal mathematical tools for program development. Despite Dijkstra's optimism that formal specifications and proof techniques will be realized within fifty years, however, many with real experience in programming-at-large continue to have their misgivings about his statement. The problem is: to what extent formal mathematical techniques will help if the user is not sure of his own requirements.

Only time will settle the controversies brought to the fore by this debate.

Srinivasarao Damerla
Department of EECS
University of Illinois at Chicago
Chicago, IL 60680

Viewpoint (*continued from p. 392*)

there should be some means of removing old features that have more efficient or more reliable replacements. We decided to create a category of these "obsolete features," which would be listed in the Fortran 90 standard, possibly be removed from the Fortran 2000 standard, and possibly fade from compilers and general use around 2020. This schedule was considered too radical by many people, both on and off X3J3, so all the features that really can make a difference in performance, such as EQUIVALENCE, were removed from the list, leaving things like the assigned GOTO and the PAUSE statements. Postponing the elimination of these features, in my opinion, is a great disservice to future Fortran programmers; it would have allowed for the evolution toward the "simple design based on a minimal set of essential concepts" that Hunter and others know is desirable. Surely, 30 years is sufficient time to eliminate these features from working programs when it is remembered that Fortran itself is only a little more than 30 years old; but those who still feel strongly about this one will have to try to do something about it during the next revision cycle if there is one.

The changes to Fortran, proposed in the first draft published for public review in 1987, took almost ten years to develop. However, when seen for the first time and all at once, the changes seem to many people to be more extensive than they would like. At the same time, many new members of X3J3 who had not participated in the development effort had the same reaction. All of this resulted in the very unusual meeting in 1988 that Geoffrey Hunter attended. At that meeting, some were proposing a radical reduction of features and some were proposing that we start over again. The committee was divided to such an extent that no single plan had sufficient support. Five of us, fearing that all the work of the past ten years would go down the drain, produced a proposed draft based on the then current draft, public comments, and the known wishes of the international Fortran community (for example, provision was made for multibyte characters, a feature requested by the Japanese, and square brackets were eliminated, as requested by the Europeans). This document was accepted by WG5, the ISO Fortran working group, and subsequently the features in it formed the basis of the current draft.

It is my opinion that the inclusion of new features such as array processing, modules, data structures, recursion, pointers, and enhanced procedure-calling mechanisms will mean that Fortran 90 will continue to be the most popular language for scientific and engineering computing, but of course, it is the users and their managers who will decide this.

Walt Brainerd
Director of Technical Work
ANSI X3J3
President, Unicorp Inc.
2002 Quail Run Drive N. E.
Albuquerque, NM 87122